Hacking: Uma Introdução ao *Exploiting* de Arquivos Executáveis

Thiago Peixoto

Universidade Federal de Alagoas
Instituto de Computação

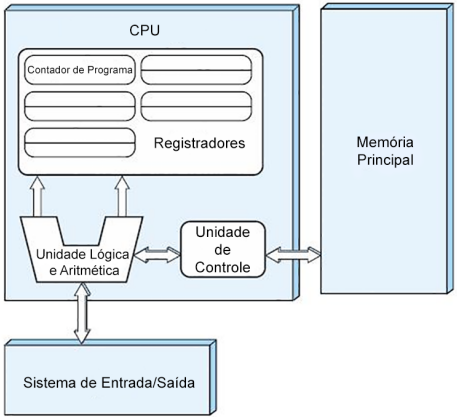
18 de setembro de 2016

Sumário

Sumário

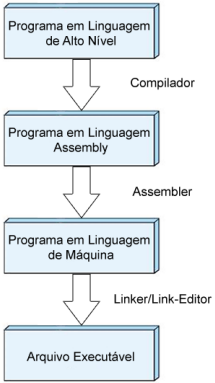
Arquitetura de Computadores

A Arquitetura de Von Neumann



Arquitetura de Computadores

O Conceito de Abstração




```
int temp = *a;
*a = *b;
*b = temp;
```

```
mov eax,DWORD PTR [rdi]
mov edx,DWORD PTR [rsi]
mov DWORD PTR [rdi],edx
mov DWORD PTR [rsi],eax
```

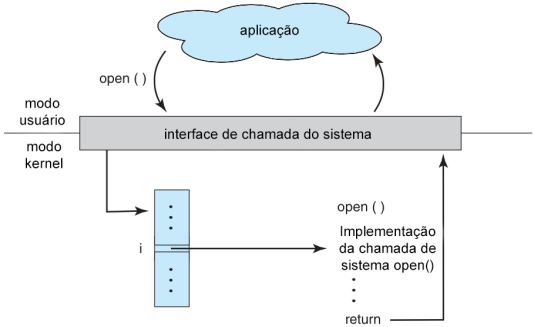
```
8B 07 8B 16 89 17 89 06
```

Sumário

Sistemas Operacionais

Chamadas de Sistema

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



Sumário

Segmentação de Memória

A memória de um programa compilado é dividida em 5 segmentos:

- **text**

  Instruções de linguagem de máquina.
- **data**

  Variáveis globais e estáticas inicializadas.
- **bss**

  Variáveis globais e estáticas não inicializadas.
- **heap**

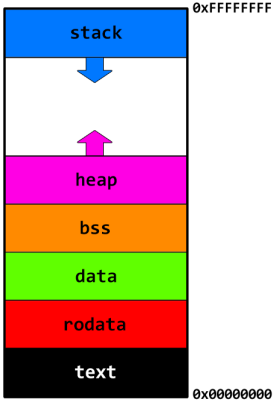
  Blocos de memória alocados dinamicamente.
- **stack**

  Variáveis locais de uma função e contexto durante chamadas de função.

Segmentação de Memória

```
char *string  = “Hello, World!”;
int size;

char *f(void)
{
    char *ptr;
    scanf(”%d”, &size);
    ptr = malloc(size);
    return ptr;
}
```



Segmentação de Memória

```c
#include <stdlib.h>

void main(void)
{
        exit(10);
}
```

```
section .text
global _start
_start:
        mov rax, [sys_exit]
        mov rdi, [ret_value]
        syscall
section .data
ret_value:  dq 0x0A
sys_exit:   dq 0x3C
```

Sumário

Sumário

Arquitetura Intel

Sumário

Registradores de Uso Geral

Sumário

Assembly

Duas opções de sintaxe:

- **AT&T**
  - instrução fonte, destino
  - mov %eax, %ecx
  - "move o conteúdo de eax para ecx"
- **Intel**
  - instrução destino, fonte
  - mov eax, ecx
  - "move o conteúdo de ecx para eax"

Assembly

Duas opções de sintaxe:

- **AT&T**
- **Intel**

Sumário

Avulsas

NOP

- Incrementa o registrador EIP para apontar para a próxima instrução;
- Alias para **xchg eax, eax**;
- Formato da instrução:

  nop

```
section .text
global _start
_start:
        nop
        mov rax, [sys_exit]
        mov rdi, [ret_value]
        syscall
section .data
ret_value:  dq 0x0A
sys_exit:   dq 0x3C
```

Movimentação de Dados

MOV

- Movimentação de dados, no formato **mov destino, fonte**;
- Formato da instrução:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov eax, [array]
        mov dword [array + 4], 0x0E
        mov ebx, 2
        mov eax, [ebx * 4 + array]
        mov edx, array
        mov eax, [edx + ebx * 4 + 4]
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
array: dd 0x0A, 0x0B, 0x0C, 0x0D
```

Movimentação de Dados

XCHG

- Troca de dados entre dois operandos, no formato **xchg op1, op2**;
- O processador usa o sinal LOCK quando um dos operandos estiver na memória;
- Formato da instrução:

  xchg reg/mem, reg
  xchg reg, mem

```
section .text
global _start
_start:
        mov eax, 0x09
        mov ebx, 0x01
        xchg eax, ebx
        xchg ebx, [num]
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dd 0x0A
```

Aritméticas

ADD

- Adição de dois valores inteiros, no formato **add destino, fonte**;
- Observar a CARRY FLAG na soma de inteiros sem sinal e a OVERFLOW FLAG na soma de inteiros com sinal;
- Formato da instrução:

  add reg/mem, imm
  add reg/mem, reg
  add reg, mem

```
section .text
global _start
_start:
        mov ecx, 0x01
        mov ebx, 0x02
        add ebx, ecx
        add ebx, [num]
        add [num], ebx
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dd 0x07
```

Aritméticas

ADC

- Adição de dois valores inteiros junto com o valor contido na CARRY FLAG de um ADD prévio, no formato **adc destino, fonte**;
- Formato da instrução:

  adc reg/mem, imm
  adc reg/mem, reg
  adc reg, mem

```
section .text
global _start
_start:
        mov al, 0xFF
        mov ah, 0x00
        mov bl, 0xFF
        mov bh, 0x00
        add al, bl
        adc ah, bh
        mov rax, 0x3C
        xor rdi, rdi
        syscall
```

Aritméticas

SUB

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov ecx, 0x01
        mov ebx, 0x0F
        sub ebx, ecx
        sub ebx, [num]
        sub ebx, 0x01
        sub [num], ebx
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dd 0x04
```

Aritméticas

SBB

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov al, 0x7F
        mov ah, 0x00
        mov bl, 0xFF
        mov bh, 0x00
        sub al, bl
        sbb ah, bh
        mov rax, 0x3C
        xor rdi, rdi
        syscall
```

Aritméticas

INC

- Incrementa o conteúdo do operando em 1, no formato **inc op**;
- Não afeta a CARRY FLAG;
- Formato da instrução:

  inc reg/mem

```
section .text
global _start
_start:
        mov ebx, 0x0A
        inc ebx
        inc dword [num]
        mov rax, 0x3c
        xor rdi, rdi
        syscall
section .data
num: dd 0x07
```

Aritméticas

DEC

- Decrementa o conteúdo do operando em 1, no formato **dec op**;
- Não afeta a CARRY FLAG;
- Formato da instrução:

  dec reg/mem

```
section .text
global _start
_start:
        mov ebx, 0x0A
        dec ebx
        dec dword [num]
        mov rax, 0x3c
        xor rdi, rdi
        syscall
section .data
num: dd 0x07
```

Aritméticas

MUL

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov rax, 0x0A
        mov rbx, 0x03
        mul rbx
        mul qword [num]
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dq 0x02
```

Aritméticas

IMUL

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov rax, 0x0A
        mov rbx, 0x03
        imul rbx
        imul rax, [num]
        imul rax, rax, 0x02
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dq 0x02
```

Aritméticas

DIV

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov eax, 0x64
        mov ebx, 0x04
        mov edx, 0x00
        div ebx
        div dword [num]
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dd 0x05
```

Aritméticas

IDIV

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov eax, [num]
        mov ebx, 0x02
        mov edx, 0xFFFFFFFF
        idiv ebx
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dd 0xFFFFFFF8
```

Deslocamento de Bits

SHL/SAL

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov al, 0xFF
        shl al, 1
        mov cl, 0x02
        sal al, cl
        mov rax, 0x3C
        xor rdi, rdi
        syscall
```

Deslocamento de Bits

SHR

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov al, 0x10
        mov cl, 0x03
        shr al, cl
        mov rax, 0x3c
        xor rdi, rdi
        syscall
```

Deslocamento de Bits

SAR

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov al, 0xF0
        mov cl, 0x03
        sar al, cl
        mov rax, 0x3C
        xor rdi, rdi
        syscall
```

Deslocamento de Bits

ROR

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov al, 0x8F
        ror al, 0x03
        mov rax, 0x3c
        xor rdi, rdi
        syscall
```

Deslocamento de Bits

ROL

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov al, 0x8F
        mov cl, 0x03
        rol al, cl
        mov rax, 0x3C
        xor rdi, rdi
        syscall
```

Lógicas

NOT

- NOT bit a bit;
- Formato:

  not reg/mem

```
section .text
global _start
_start:
        mov al, 0x0F
        not al
        mov rax, 0x3C
        xor rdi, rdi
        syscall
```

Lógicas

AND

- AND bit a bit;
- Formato:

  and reg/mem, imm
  and reg/mem, reg
  and reg, mem

```
section .text
global _start
_start:
        and dword [num], 0x03
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dd 0x07
```

Lógicas

OR

- OR bit a bit;
- Formato:

  or reg/mem, imm
  or reg/mem, reg
  or reg, mem

```
section .text
global _start
_start:
        mov eax, 0x03
        or eax, dword [num]
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dd 0x07
```

Lógicas

XOR

- XOR bit a bit;
- Formato:

  xor reg/mem, imm
  xor reg/mem, reg
  xor reg, mem

```
section .text
global _start
_start:
        mov eax, 0x03
        xor eax, dword [num]
        mov rax, 0x3C
        xor rdi, rdi
        syscall
section .data
num: dd 0x07
```

Controle de Fluxo

JMP

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        jmp label
        mov rbx, 0x01
        mov rcx, 0x02
label:
        mov rax, 0x3C
        xor rdi, rdi
        syscall
```

Controle de Fluxo

J[CONDIÇÃO]

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

```
mov reg/mem, imm
mov reg/mem, reg
mov reg, mem
```

Controle de Fluxo

LOOP

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov ecx, 0x0A
label:
        add dword [num], ecx
        loop label
        xor rdi, rdi
        mov rax, 0x3C
        syscall
section .data
num: dd 0x00
```

Controle de Fluxo

CALL

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
extern printf
section .text
global _start
_start:
        mov rdi, string
        call printf
        xor rdi, rdi
        mov rax, 0x3C
        syscall
section .data
string: db "Uma string!", 0x0A, 0x00
```

Controle de Fluxo

INT

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov eax, 0x04
        mov ebx, 0x01
        mov ecx, string
        mov edx, tamanho
        int 0x80
        xor ebx, ebx
        mov eax, 0x01
        int 0x80
section .data
string: db "Uma string!", 0x0A
tamanho: equ $ - string
```

Controle de Fluxo

SYSCALL

- Movimentação de dados no seguinte formato: **mov destino, fonte**;
- Formato:

  mov reg/mem, imm
  mov reg/mem, reg
  mov reg, mem

```
section .text
global _start
_start:
        mov rax, 0x01
        mov rdi, 0x01
        mov rsi, string
        mov rdx, tamanho
        syscall
        xor rdi, rdi
        mov rax, 0x3C
        syscall
section .data
string: db "Uma string!", 0x0A
tamanho: equ $ - string
```

Sumário

Stack Overflow